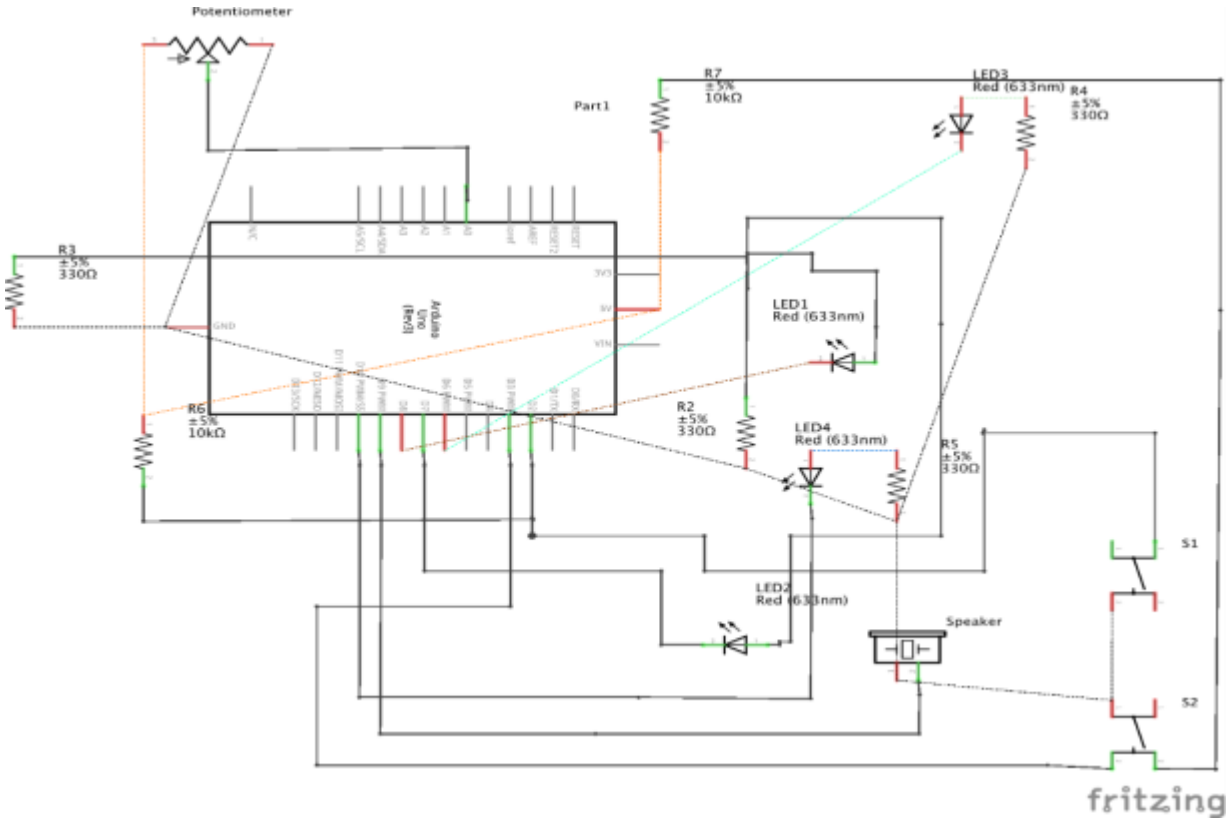
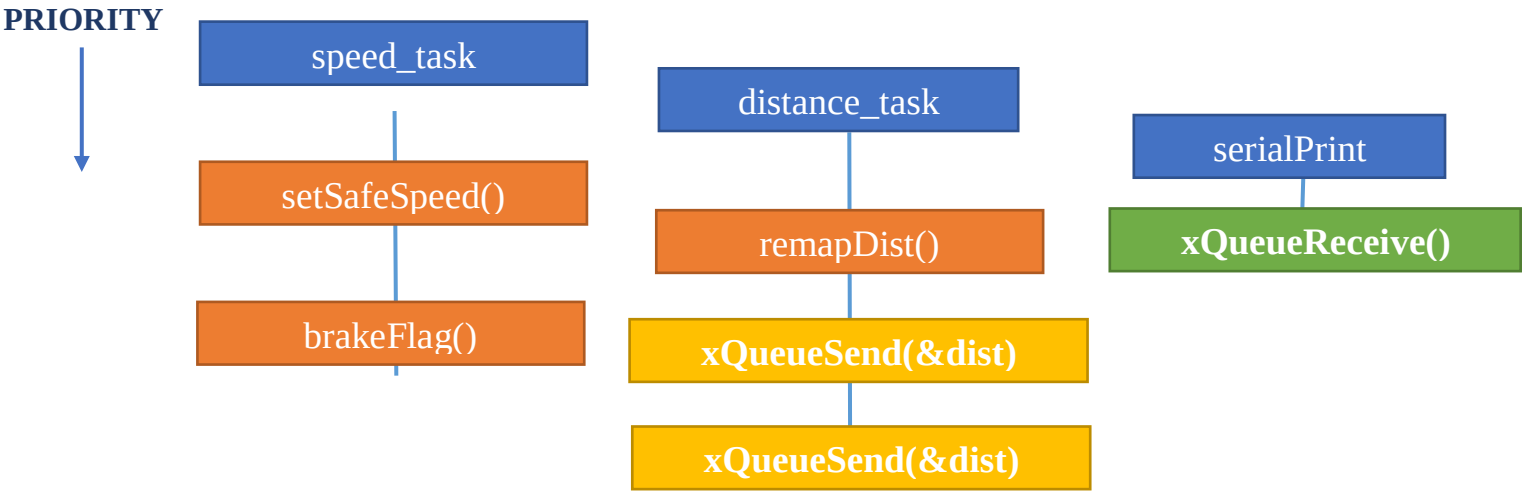
CG2271 Real Time Operating Systems
Lab 6
Answer Book

Name: Baghabrah Dana	Matric Number: A0144902L
Name: Anniya Baskaran	Matric Number: A0141812R

The purpose of Lab 6 assignment is to design the controller of a car with an automated driving system.



Device	**Pin**
ledY1 (yellow)	6
ledY2 (yellow)	7
ledY3 (yellow)	8
ledBrake (red)	9
PTTM	A0
PUSH_B1	2
PUSH_B2	3



In this assignment, we used FreeRTOS to create tasks and used message passing queues to support communication and synchronization between tasks. The potentiometer was used to emulate the distance from the vehicle in front. As an ADC conversion is required for the potentiometer readings, a remapDist() function was used to convert the analog values which has a range from 0 to 1023 to digital values which have a range of 0 to 255.

```
//remap potentiometer from [0-1023] to [0-255]
int remapDist(int value){
	return (int) (value * 255.0 / 1023.0);
}
```

Using freeRTOS, 3 tasks were created in this assignment and these are the speed_task, distance_task and serialPrint. The speed_task sets the lighting of the LEDs according to the current speed that is set according to the distance in setSafeSpeed().

```
//decreases current speed after safe brake and gets the minimum value in
the pair of desired speed and safe speed
void setSafeSpeed(int dist){
	if( dist == 1){
		current_speed = 0;
	}
	else if(dist == 2){
		if(desired_speed > 0)
			current_speed = 1;
	}
	else if(dist == 3){
		if(desired_speed == 3)
			current_speed = 2;
	}
	else {
		if(desired_speed < 3){
			current_speed = desired_speed;
		}
		else{
			current_speed = 3;
		}
	}
}
```

The code above implements the speed update based on this equation.
current speed = minimum (driver desired speed setting, safe speed setting at current distance)

desired_speed is based on the push buttons clicks by the user. Initially, based on the safe distance, we set the current_speed as the safe speed. The safe speed is determined based on the distance as shown in the table from the Lab6 manual. However, if the current speed is more than the desired speed, we set the current speed as the desired speed.

When the current speed is 0, none of the yellow LEDs lights up since 0 being the lowest speed is indicated by 0 yellow LEDs lighting up. Moreover, the brake is not activated then so the red LED does not light up. As the purpose of the buzzer is to emulate the engine behaviour, a duration of 1519 is set for the buzzer tone to keep the volume of the buzzer as low as possible.

```
if (current_speed == 0){
      digitalWrite(ledY1, LOW);
      digitalWrite(ledY2, LOW);
      digitalWrite(ledY3, LOW);
      digitalWrite(ledBrake, LOW);
      tone(SPEAKER, 1519);
}
```

When the current speed is 1, one of the yellow LEDs lights up. In this case, only the yellow LED connected to digital pin 6 lights up. A duration of 1432 is set for the buzzer tone to increase the volume of the buzzer by a note. Unless the safe distance is less than 2, the brake will not be activated meaning red LED will not light up. If the safe distance is less than 2, brake will be activated by setting brake flag as 1. The safe speed is then set again according to the safe distance in the setSafeSpeed().
The brakeFlag() is then called which mean the red LED will now light up.

```
//indicates safe brake
void brakeFlag(void){
      TickType_t xCurrTime = xTaskGetTickCount();
      if (brake_flag == 1){

            if (xCurrTime - previousTime >= 1000){
                  previousTime = xCurrTime;

                  digitalWrite(ledBrake, HIGH);

            }
            brake_flag = 0;
      }
}

//turns off the RED led
digitalWrite(ledBrake, LOW);

//Checks if safe distance is less than 2, while speed is 1
      if((safeDistance < 2) && (current_speed == 1)){
            brake_flag = 1;
            digitalWrite(ledBrake, LOW);
            brakeFlag();
      }
```

```
      …

//Updates current speed depending on the safe distance
setSafeSpeed(safeDistance);

//lights the required LEDs and activates the speaker
else if (current_speed == 1){

      digitalWrite(ledY1, HIGH);
      digitalWrite(ledY2, LOW);
      digitalWrite(ledY3, LOW);
      tone(SPEAKER, 1432);

}
```

When the current speed is 2, two of the yellow LEDs light up. In this case, the yellow LEDs connected to digital pins 6 and 7 light up. A duration of 1136 is set for the buzzer tone to increase the volume of the buzzer by a note. Unless the safe distance is less than 3, the brake will not be activated meaning red LED will not light up. If the safe distance is less than 3, brake will be activated by setting brake flag as 1. The safe speed is then set again according to the safe distance in the setSafeSpeed(). The brakeFlag() is then called which mean the red LED will now light up.

```
else if (current_speed == 2){

      digitalWrite(ledY1, HIGH);
      digitalWrite(ledY2, HIGH);
      digitalWrite(ledY3, LOW);
      tone(SPEAKER, 1136 );

}
```

When the current speed is 3, all the yellow LEDs light up. A duration of 956 is set for the buzzer tone to increase the volume of the buzzer by a note. Unless the safe distance is less than 4, the brake will not be activated meaning red LED will not light up. If the safe distance is less than 4, brake will be activated by setting brake flag as 1. The safe speed is then set again according to the safe distance in the setSafeSpeed(). The brakeFlag() is then called which mean the red LED will now light up.

```
else if (current_speed == 3){
       digitalWrite(ledY1, HIGH);
       digitalWrite(ledY2, HIGH);
       digitalWrite(ledY3, HIGH);
       tone(SPEAKER, 956);
}
```

The speed_task is set as the task with highest priority in loop(). Since it containes the brake flags function calls.

To set the distance to different ranges of values of the potentiometer, the following was done in the distance_task() function.

```
value = remapDist(value);
if (value <= 64){
      distance = 1;
}
else if ((value > 64) && (value <= 128)){
```

```
	distance = 2;
}
else if ((value > 128) && (value <= 192)){
	distance = 3;
}
else if (value > 192){
	distance = 4;
}
```

The distance_task is set as the task with medium priority in loop().

The speed_task, distance_task and brake_flag have a period of 1000ms (1 second). According to the requirements and to support synchronization.

The serialPrint() print the necessary information on UART. Since it has to only print the statements, it is set as the task with lowest priority in loop().

```
//Prints information on UART
void serialPrint(void *p){
	while(1){
		int taskDistance;
		xQueueReceive(xQueueUART, &taskDistance, portMAX_DELAY);
		Serial.print("Desired Speed: ");
		Serial.println(desired_speed);
		Serial.print("Current Speed: ");
		Serial.println(current_speed);
		Serial.print("Distance: ");
		if( taskDistance != 1)
			Serial.print(taskDistance);
		Serial.println("d");
	}
}
```

Two message queues were created in this assignment. xQueueUART is created to send information to the communication task (Serial Print Task). xQueueSafeDist is created to update the safe distance.

```
//Sends safe distance to speed Task
xQueueSend(xQueueSafeDist, &distance, portMAX_DELAY);

//Sends safe distance to serialPrint
xQueueSend(xQueueUART, &distance, portMAX_DELAY);
```

We used portMAX_DELAY since we want the queue to keep waiting until it finds one empty space, rather than returning an error if it did not find.

Two interrupt service routines were created to detect the pressing of the push buttons. When PUSH_B1, a pushbutton connected to digital pin 2, is pressed, int0ISR() is called. int0ISR increments the current speed if it is less than 3 and set desired speed to be the same as the current speed. When PUSH_B2, a pushbutton connected to digital pin 3, is pressed, int1ISR() is called. int1ISR decrements the current speed is more than 0 and set desired speed to be the same as the current speed. Both ISR have a debouncing delay of 500ms.

Rate Monotonic Scheduling (RMS) Policy was used in this assignment to schedule the 3 tasks. Polling the message queues allow distance_task to check for a signal.